CSC 413 Term Project Documentation

Summer 2024

Student name: Guiran liu

Student ID: 923620812

Class : CSC413-02

GitHub Repository Link:

https://github.com/csc413-SFSU-SU2024/interpreter-GuiG2023

Details

Each student must write documentation for their term project. This document will cover the entire term project from beginning to end.

Your documentation MUST contain the following sections:

1. Title page containing
    1. Student's Name
    2. Class, Semester
    3. A Link to your repository.
2. Introduction
    1. Project Overview (the focus of the term project)
    2. Introduction of the Tank game (general idea)
3. Development environment.
    1. The version of Java Used
    2. IDE Used
    3. Were there any special libraries or special resources where you got them from and how to install them.
4. How to build or import your game in the IDE you used.
    1. Note saying things, like hitting the play button and/or clicking import project, is not enough. You need to explain how to import and/or build the game.
    2. List what Commands that were running when building the JAR. Or Steps taken to build jar.
        1. These can be the steps done either at the command line or in IntelliJ.
    3. List commands needed to run the built jar
        1. These can be the steps done either at the command line or in IntelliJ.
    4. How to run your game. As well as the rules and controls of the game.
    5. Assumptions Made when designing and implementing your game.
    6. Tank Game Class Diagram
    7. Class Descriptions of classes implemented in the Tank Game
        1. No need to over-explain but state the purpose of each class.
    8. Self-reflection on the Development process during the term project
    9. Project Conclusion.

When completing this documentation please make sure you are concise but that each section contains enough information. Point penalties will be added for insufficient information or missing sections.

The final documentation ***MUST*** be submitted in PDF FORM. Please submit your final documentation in PDF form to Canvas by the deadline at the heading of this section (note the repetition, must be important). ***Submitting documentation that is not a PDF will cause a 10-point penalty.***

1 Introduction

1.1 Project Overview

This project consists of a 2D game written in Java, a presentation, and final documentation for the term project. The goal of this term project is to practice good OOP. We will make a Tank War game, just like the classic tank game -- Battle City, almost building and programming the game from scratch.

1.2 Introduction to my tank game

My tank game has two players, and these two play against each other; the one who eliminates the other one is the winner of the game. In my game, I have three different kinds of power-ups they are speed power enhanced shooting power-ups, and life you will get benefits when you pick up each of those.

About the environmental objects in my game, I have two different unbreakable walls. The one made of concrete color is white is the border of the whole game, and the orange one, there's another unbreakable wall, but actually, it could be broken if you add on ten times and also I have a brick ball wall which could be broken by shooting or by tank crash.

I also have sand and rivers. If you get into the sand, your movement will get extremely slow and easily be aimed by others. Even if you get out of this sand, your speed cannot be restored, so you have two ways to restore your original speed. One way is that you pick up speed power up, then you can not only restore to the original speed but can also enhance your speed. The 2nd way is you can get into the river to wash your sand. When you get out of the river, you will find your speed becomes the original speed, and the river will carry you some distance when you inside the river because the river always has river flows, it will carry you automatically even if you release your keyboard, that make my game more interesting.

The most creative thing I generated in my game is I randomly created enemy Forts that are inherited from the player tank class but cannot move. They will rotate to the player tank, and should player tanks cool down, time is longer than players shot cool down time so you can eliminate the enemy forts, and players can also use the enemy force to get benefit from that. For example, let the Forts rotate to you, and you run away. The bullet will shoot the other players. That's the way you could benefit from that. That makes my game more interesting and challenges each tank, and enemy forts have three lives for player tanks. If you lose one life, you will be reset to the starting point, but no worries, each life has four chances to be shot.

2 Development Environment

1. OpenJDK 21.0.2 2024-01-16

   OpenJDK Runtime Environment (build 21.0.2+13-58)

   OpenJDK 64-Bit Server VM (build 21.0.2+13-58, mixed mode, sharing)

2. IDE: IntelliJ IDEA 2023.3.3 (Ultimate Edition)

3. Java Swing / Java AWT (Not special library)

4 How to Build/Import your Project

4.1 Note saying things, like hitting the play button and/or clicking import project, is not enough. You need to explain how to import and/or build the game.

1.My game is easy to build, I pushed all my clothes and resources in my GitHub repository, so if anyone wants to download my game, clone my game from my GitHub repository and open it with intellij, and open it as a project, double check all the pictures sound animations set it as resources in this way we finish bud the game.

2.Go to jar folder, download the csc413-tankgame-GuiG2023.jar to your local folder(any), use terminal to that folder(cd), then use java -jar csc413-tankgame-GuiG2023.jar

4.2 List what Commands that were running when building the JAR. Or Steps taken to build jar. These can be the steps done either at the command line or in IntelliJ.

After Colne all the resources and codes from my GitHub repository, I have to set the tank game----simple ----resources as the dependency of this project because I had already put all of the BGM animations, pictures and sounds in there also we can set tankgame folder as resource 2 in this way we could keep all my program can use all the resource that I created before if we finish doing that we can run my game in the next step

4.3 List commands needed to run the built jar. These can be the steps done at the command line or in IntelliJ.

Generally speaking, I only use Intelli J and find my launcher class selected, and click run the program but there's a way to use bash to play my game. I will test it later and try to write some pseudo commands here： cd path/to/jar 2. java launcher(not sure yet, will test then)

(updated)Go to jar folder, download the csc413-tankgame-GuiG2023.jar to your local folder(any),
use terminal to that folder(cd), then use java -jar csc413-tankgame-GuiG2023.jar

4.4 How to run your game. As well as the rules and controls of the game.

Basically, if someone has already cloned all my resources and set up all the settings in IntelliJ you can find my launcher class and click the primary method to run my game.

Player Tank Control:

P1: Use 'WASD' keys to move the tank, and 'Space' to shoot.

P2: Use arrow keys (Up/Down/Left/Right) to move the tank, and 'P' to shoot.

Game Terrain Introduction:

- Impassable Concrete Walls: These are located at the map's edges and cannot be breached. - Indestructible Walls: These require ten continuous hits for destruction.

- Destructible Walls: Can be destroyed by one hit or by ramming, though ramming causes a brief halt to your tank.

- Sand: Tanks slow down when entering sand and do not immediately regain speed upon exiting. Speed can be increased by picking up a speed power-up or entering rivers for cleaning. - Rivers: Flowing river water will carry the tank slowly in the direction of the flow, even if no movement keys are pressed.

Power-Ups:

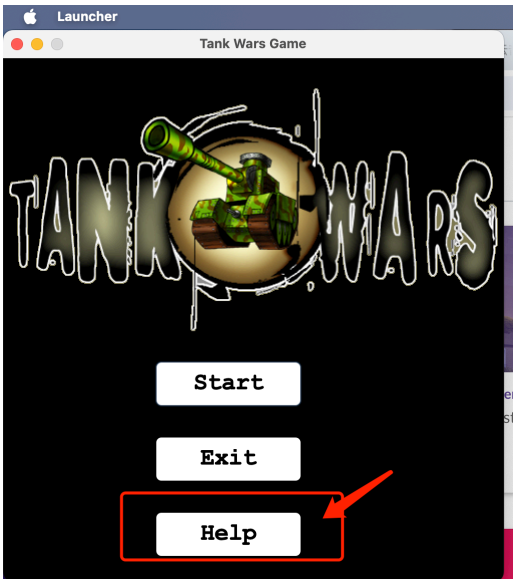
- Shooting Boost: Enables shooting in multiple directions simultaneously for 30 seconds.

- Speed Boost: Increases tank speed to its maximum.

- Extra Life: Grants an additional life. Enemy Bunkers:

enemy forts

- Multiple enemy forts are generated at random locations at the start of each game. They are stationary but will rotate towards the player's tank and attack, and they can be destroyed.



4.5 Assumptions Made when designing and implementing your game.

Technical Assumptions

Platform Compatibility: The game is assumed to run on PCs with IntelliJ installed, and it is expected to have sufficient processing power and graphics support for smooth gameplay.

Network Requirements: None

Player Behavior Assumptions

Game Learning Curve: It is assumed that players will be able to quickly learn and adapt to the game's controls and mechanics.

Player Engagement: It is assumed that players are interested in tactical and strategy games and are willing to invest time to master the game.

Game Design Assumptions

terrain Impact: It is assumed that the different terrains on the game map, such as rivers, sand, and concrete walls, will significantly affect tank movement and tactical choices.

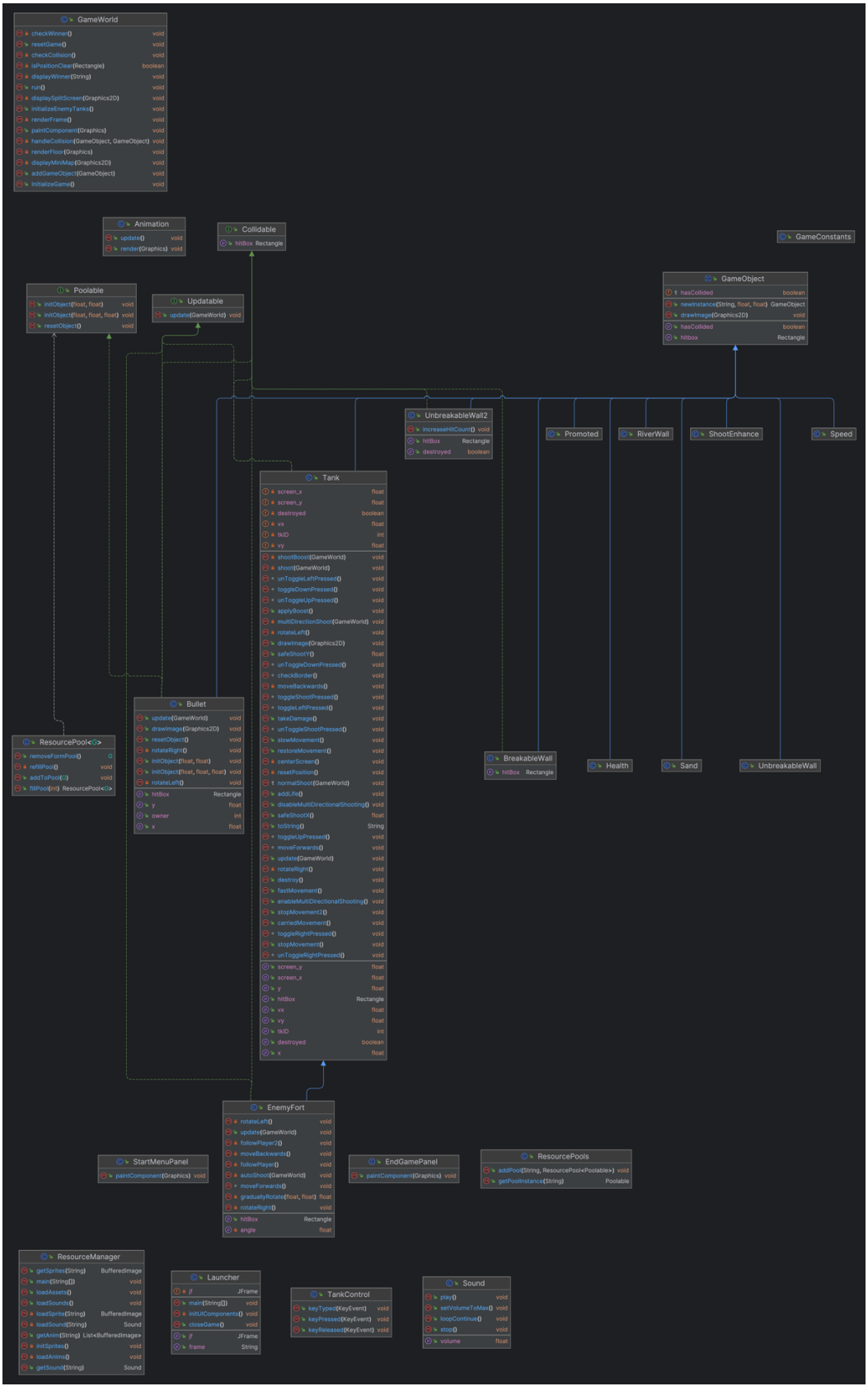
Market Assumptions

Target Audience: The game primarily targets those who reminisce about the classic Battle City and those with relevant gaming experience.

Game Balance Assumptions

Balancing Mechanism: It is assumed that the game's balance will be adjusted based on testing and player feedback to ensure no tank or strategy is overly powerful or weak (temporarily).

4.6 Tank Game Class Diagram



4.7 Class Descriptions of classes implemented in the Tank Game

Animation: Manages animation frames for visual effects like explosions or other dynamic visuals within the game, it also control the update of animation and render the animation

BreakableWall: Represents walls in the game that can be destroyed by tank fire, changing the landscape of the game environment when destroyed.

Bullet: Represents projectiles fired by tanks. Usually,This class handles bullet physics, collision detection, and rendering bullets on the screen but I put all of them in game world class to handle the different kinds of collision maybe it's not a perfect way but it does work also we have a get owner and set owner to identify which player tank's bullet.

Collidable: An interface that defines objects that can participate in collisions, typically implemented by objects like tanks, walls, and bullets.

EnemyFort: this class inherits from tank class, A type of enemy tank that may have more properties or behaviors than regular player tanks, but for the balance of the game it cannot move.

GameObject: A base class for all objects in the game that can appear on the game board, including tanks, walls, and bullets, game object class control all the different kinds of game objects and all the basic functions that the children class may have is also necessary for creating the map.

GameWorld: Manages the main game loop, rendering, and updates the state of the game. This class is central to game operations, handling the initialization and ongoing management of the game environment.

Health: Likely represents health packs that tanks can pick up to restore health.

Poolable: An interface for objects that can be managed in an object pool, improving performance by reusing objects instead of creating new ones.

Promoted: This could be related to a special status or power-up that changes the properties or abilities of a tank(Enhanced shooting).

ResourcePool: Manages pools of resources, such as bullets or explosions, to optimize performance by reusing objects.

RiverWall, Sand, ShootEnhance, Speed: Various classes representing different types of environmental features or power-ups that affect gameplay, such as obstacles that affect tank movement or items that modify shooting mechanics.

Sound: Manages audio effects within the game, handling the loading, playing, and stopping of sound files.

Tank: Represents player-controlled or AI-controlled tanks within the game. Includes methods for movement, shooting, and collision handling.

TankControl: Handles input from the player, translating keyboard or mouse input into game actions like moving or firing.

UnbreakableWall and UnbreakableWall2: Represent walls that cannot be destroyed(kind of), serving as permanent obstacles within the game environment.

Updatable: An interface that objects can implement if they need to be updated every game tick, such as moving or checking for collisions(not fully implemented in my game).

EndGamePanel: A user interface panel displayed at the end of the game, offering options like restarting or exiting.

StartMenuPanel: The initial menu is shown to players, providing options to start the game or exit.

GameConstants: （provided by prof）This class defines various constants used throughout the game, such as screen sizes, dimensions of the game world, and unit sizes. It provides standardized measurements that ensure different parts of the game work together cohesively.

Launcher: （provided by prof）The Launcher class initiates the game's main window and user interface. It manages different game screens (like the start menu, game interface, and end game panel) using a CardLayout to handle visibility and transitions between screens. The Launcher also acts as the starting point for the game loop, running it on a separate thread to ensure smooth UI responsiveness.

ResourceManager: This class manages all resources, such as images and sound files, needed by the game. It typically contains methods for loading these resources to ensure they are available throughout the game, which helps optimize performance and memory usage by avoiding redundant resource loading.

ResourcePools: （provided by prof）The ResourcePools class handles the management of resource pools, especially for objects frequently created and destroyed, like bullets. By using the object pool pattern, this class reduces runtime memory allocation and garbage collection, thereby enhancing performance. Each type of object can have its dedicated pool for reuse.

4.8 Self-reflection on the Development process during the term project

This has been the most exciting and practically enriching project I've ever done. Since starting to learn Java, I've only been introduced to fragmented concepts without the opportunity to apply what I've learned fully. This project was an excellent chance for me to engage with my passion for gaming and to develop a game on my own, which I've always wanted to do. I am deeply grateful to the professor for this well-structured project and for the meticulous guidance provided throughout the course. With this guidance, I developed this simple yet incredibly satisfying game independently.

During the development process, I initially struggled with understanding which classes should exist and how they should interact to achieve the functionalities I wanted, like shooting bullets or representing the impact of a hit. The professor didn't just focus on the details but also taught us how to efficiently use inheritance and interfaces, optimize algorithms for performance, and handle dynamic updates in the UI. I repeatedly watched the lecture recordings—at least twice each—beyond attending every class and actively participating in grasping the concepts being taught. This effort greatly enhanced my understanding and enabled me to add many personal touches to the game, making it unique and well-received during our class presentation. The entire experience, including the extensive debugging due to minor oversights that initially escaped my notice, was incredibly educational. It pushed me to adapt and apply my skills creatively, culminating in a project that not only solved technical challenges but also garnered positive feedback from my peers.

4.9 Project Conclusion.

I sincerely dedicated myself to completing this project, not only meeting the basic requirements but also incorporating new elements and gameplay mechanics. My deep dive into Java through this project has not only solidified my understanding but also significantly increased my interest in computer science. Thank you, Professor Souza!